

Microcontroller Basics Course

part 4: the READS51 C compiler

Anyone who seriously intends to work with microcontrollers must sooner or later use the C programming language. In this final instalment of the Microcontroller Basics course, we use the READS51 C compiler from Rigel.

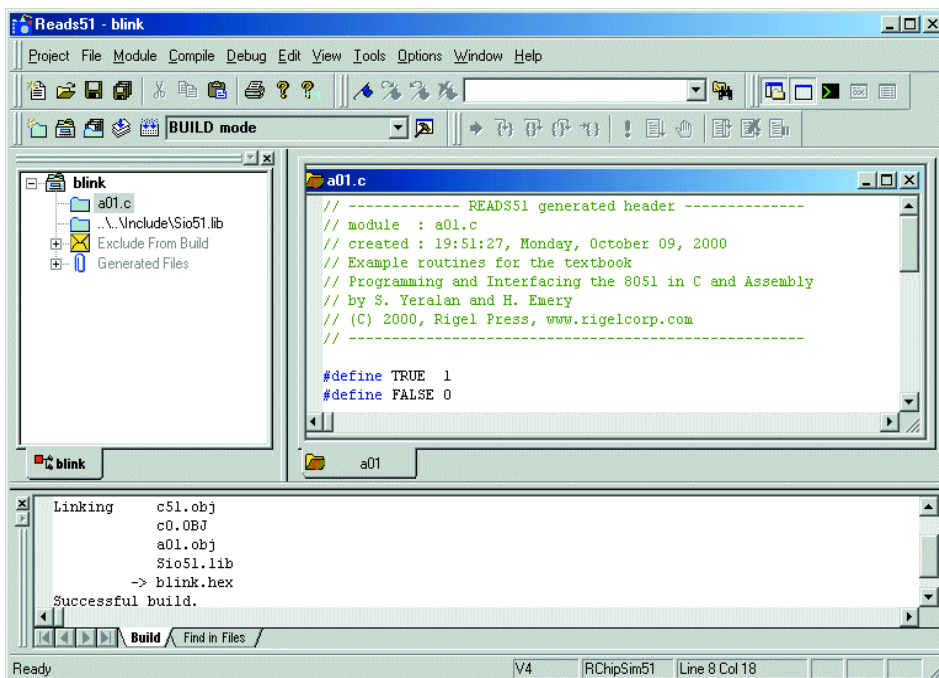


Figure 1. READS51 in action.

Up to now we have used assembler and BASIC-52 as programming languages in the Microprocessor Basics Course. Now it's time to work with C, using a compiler that can be selected and downloaded from the following Internet address:

<http://www.rigelcorp.com/8051soft.htm>

The relevant files to fetch are

[SetupReads51.exe](#) and [Reads51.pdf](#).

A C compiler translates a source text into pure machine code, in contrast to a Basic interpreter, which only generates intermediate code that must interpreted and executed at run time. C is thus many times faster than Basic.

The C language has been around for a long time and is available for many different systems. Its decisive advantage is that C programs are largely independent of the hardware

used. The results of laborious effort can thus be relatively easily ported to other systems. ANSI C was defined as early as 1988 (ANSI stands for 'American National Standards Institute') in order to create a common standard. A smaller version called 'Small C' has been specially developed for microcontroller systems. Although it has certain limitations compared with ANSI C, such as the absence of 'real' variables, it has the advantage that it can be used with very small systems. Various free compilers for Small C can be found on the Internet. For this course we have chosen READS51, since it is particularly suited to novices and has a convenient user interface.

READS51 was specially developed by Rigel for the educational market and is intended to support their microcontroller boards. The company makes this product available to anyone using it for purely private purposes or educational use. Rigel have kindly given *Elektor Electronics* permission to use the compiler for the Microcontroller Basics course. All interested readers should therefore download READS51 from Rigel's Internet site and install it on their systems. By the way, you can also find many other equally interesting help files at this site. All examples for the Microprocessor Basics course have English labels and comments. We simply couldn't

Listing 1. The first sample program.

```
// ----- READS51 generated header -----
// module : a01.c
// created : 19:51:27, Monday, October 09, 2000
// Example routines for the textbook
// Programming and Interfacing the 8051 in C and
// Assembly
// by S. Yeralan and H. Emery
// (C) 2000, Rigel Press, www.rigelcorp.com
// -----

#define TRUE 1
#define FALSE 0

#include <sfr51.h> // P1_0 is defined here
// prototypes
#include <Sio51.h>

main(){
int n;

// -- initialize serial port (9600 Baud) --
InitSerialPort0(DEF_SIO_MODE);
//DEF_SIO_MODE is defined in
<Sio51.h>
putc('\n');

// endless loop
while(TRUE)
{
P1_0=0; // LED on
putc('+');
for(n=0; n<10000; n++); // waste some cycles
P1_0=1; // LED off
putc('0');
for(n=0; n<10000; n++); // waste some cycles
}
}
```

proceed any further without using this international approach.

The best way to get started with READS51 is to use one of the accompanying examples. The project blink can be loaded from Project/Open Project. If you double-click on source text file for the main module, a01.c, the source text will appear in the Editor window (see **Listing 1**).

A C program always has a main function called `main()` that is executed when the program is started. At first glance, the sample program blink appears to contain only this function, but in fact some other functions related to the serial interface of the microcontroller are also used.

These functions are located in the module `Sio51.h`. They open the serial interface at 9,600 baud (with a crystal frequency of 11.0592 MHz), which is exactly what the *Elektor Electronics* Flash Board needs. Just in case you did not know, the highly successful 89S8252 Flash Board was described in the December 2001 issue of *Elektor Electronics*.

For C beginners, the program notation may at first seem a bit odd, so explanations of some of the details are in order:

```
#define TRUE 1
Defines a constant (TRUE will be
replaced by '1' wherever it appears).
```

```
#include <sfr51.h>
```

Links in a header file containing definitions.

```
main()
{
...
}
```

Forms the principal function `main`. All the instructions for this function are contained in a block of instructions enclosed by a pair of curly brackets.

```
int n;
Declares a variable n of type integer, whose
allowed range of values is -32768 to +32767.
A semicolon (;) terminates the line.
```

```
InitSerialPort0
(DEF_SIO_MODE);
Calls a function with a passed parameter, in
this case a function in module Sio51.h that
initialises the serial interface.
```

```
// endless loop
A comment, which increases the readability
of the program but is not translated with the
actual program.
```

```
while(TRUE)
{
...
}
```

Forms a loop. In place of `TRUE` for an endless loop, a different condition could be used here to define the condition under which the loop is to be traversed. All instructions that are to be executed in the loop are again enclosed in curly brackets.

```
P1_0=0;
An instruction. Here the bit variable P1_0 is
assigned the value '0'.
```

```
putc('+');
Text output via the serial interface. The func-
tion putc is defined in Sio51.h. A text char-
acter, which is a variable or constant of the
type char (character = text character,
always one byte), is transferred.
```

```
for(n=0; n<10000; n++);
Forms a counting loop, which would be writ-
ten in Basic as 'For n=1 to 10000: Next
n'. Here the loop does not contain any
instructions, as can be seen from the semi-
colon. A block of instructions enclosed by
curly brackets could also be located here.
```

Even if you haven't yet fully grasped all the subtleties of C programs, it's interesting to see whether this program will run on the

ATMELISP, a new download tool

The simple loader program MicroFlash.exe for downloading programs to the 89S8252 board works only with COM1 or COM2 and does not report back regarding the success of the download, which has led to problems for some users. However, *Elektor Electronics* readers do not sit idle in such situations. Ulrich Bangert (DF6JB) has consequently developed a new and significantly more extensive program named ATMELISP, which allows the Flash memory to be programmed using various types of systems. Besides the Atmel Starter Kit and a proprietary board, the program also supports the *Elektor Electronics* system and the ModuleBus system (EX52-Flash). The new software can be downloaded from the *Elektor Electronics* home page.

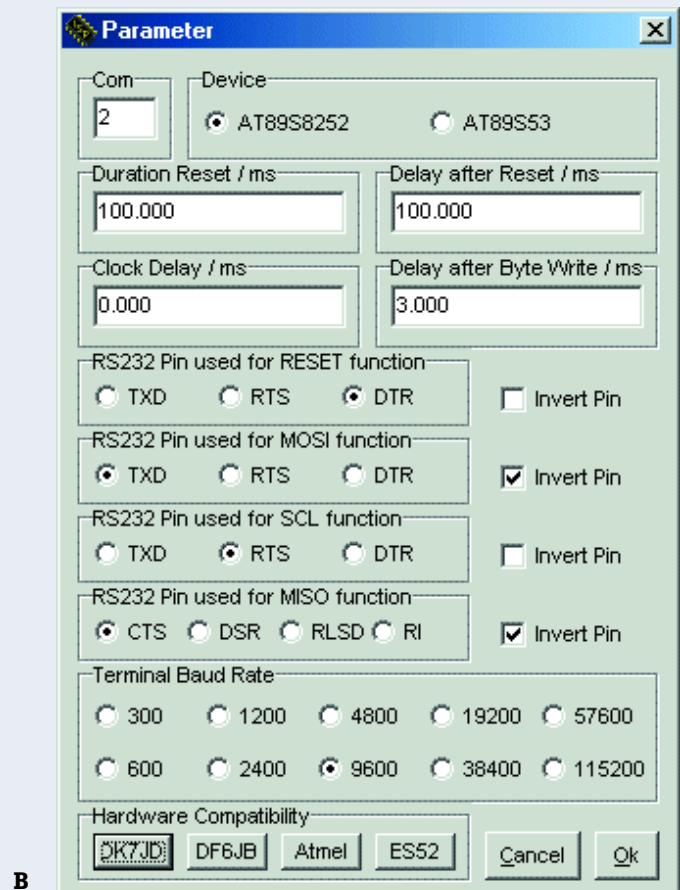
When the zip archive has been unpacked, you will have an .exe program and a comprehensive help file. The start-up screen (**Figure A**) is small and can easily be placed on the monitor next to other applications. Larger windows only appear when program functions are executed. The first thing you must do is to select the serial interface, the connected device and other critical parameters. A click on the button marked 'DK7JD' (which is B. Kainka's amateur radio call sign) configures the proper assignment of the programming lines to the RS232 lines used for the *Elektor Electronics* circuit board. Here you can also see that it is easily possible to use ATMELISP to program any desired circuit board you have developed yourself that uses the same processor, since three lines are simply selected and appropriately assigned. In some cases, it may be necessary to adjust the delay times. Our experience shows that with a relatively slow PC the value of Clock Delay must be increased from 0 to 0.01 ms. **Figure B** shows the window for selecting the configuration parameters.

The rest of the procedure can be illustrated using a concrete example. The Flash ROM of the microcontroller is to be loaded with the first sample program from the C compiler. This requires the code to first be read into the buffer. ATMELISP can read files in binary and Intel hex formats. Here the file *Blink.hex* is loaded. It has also proven to be worthwhile to have a quick look at the built-in hex editor after loading the file (**Figure C**), in order to see the content and size of the file.

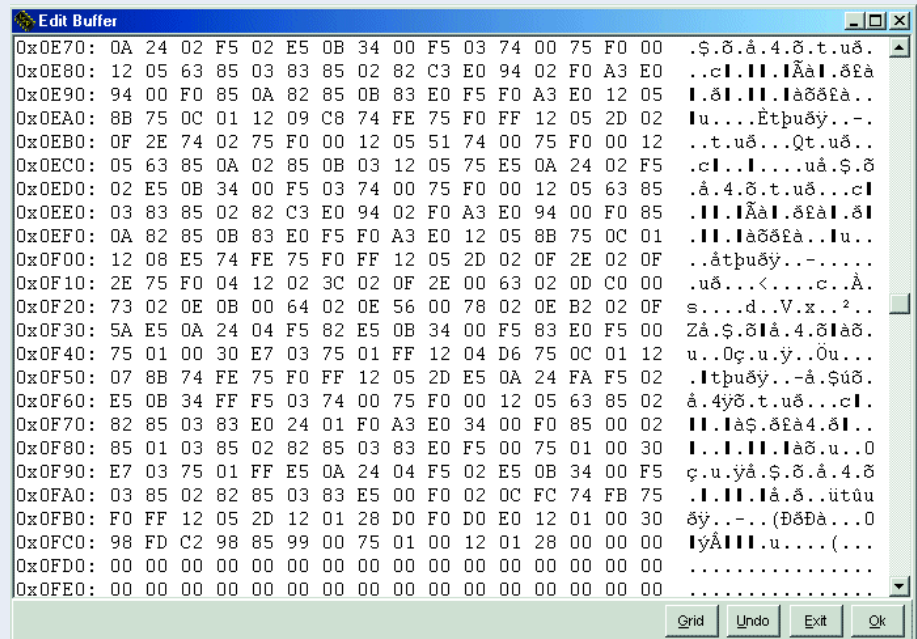
To program the microcontroller, select Device/Write Buffer to Code Memory. Here you must be careful not to confuse the Code Memory with the Data Memory, which is the 2-kbyte EEPROM data region of the microcontroller. Both of these memories can be programmed and read. Besides this, it is possible to load 'lock bits' into the microcontroller in order to prevent the loaded software from being read (**Figure D**). But be careful with the lock bits: if all three bits are set, any further serial programming of the chip is blocked! In this case, it is also no longer possible to erase the entire chip using the program. Only a parallel programming device



A



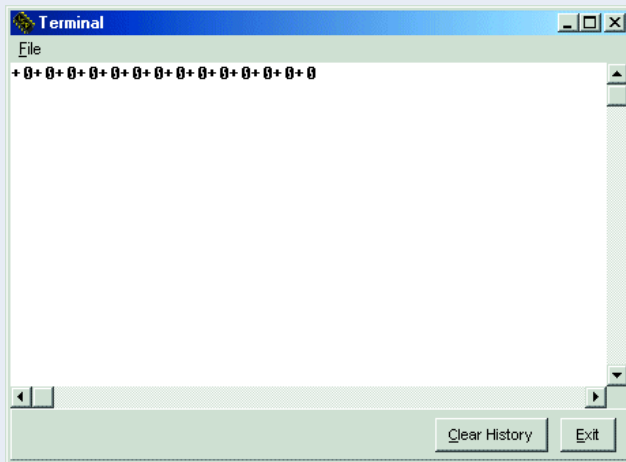
B



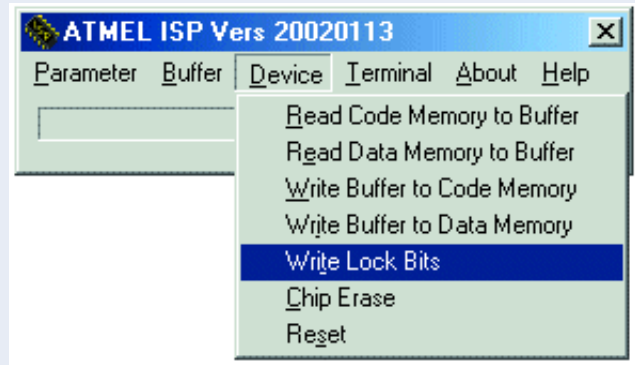
C

can get you out of this trap.

ATMELISP also includes a terminal function (**Figure E**)



D



that can be used to view the outputs from the first sample program. This requires the interface cable to be connected to the other interface socket on the board.

E

Flash Board. Before doing so, you must first compile the program, which means having it translated into machine language. The program can be translated using `Compile/build` or by simply pressing `F9`. The process is relatively complicated, since the individual object modules must first be translated, following which they are linked together to form a complete program. The final result is a file in Intel hex format named `Blink.hex`. It is located in the project directory

\work\blink and can also be seen under 'Generated files' as .hex.

The project's Intel hex file can now be simply transferred to the Flash Board using the MicroFlash program. When downloading the program into the Flash memory of the microcontroller, you will notice that in spite of the simplicity of the source text, the translated version is relatively large (4 kbytes). That is primarily due to the module C51.obj, which was linked in. This module contains all the functions

provided by READS51, including ones that are not actually necessary here.

After the program has been successfully downloaded, it's time for testing. Connect a LED and a series resistor between P1.0 and V_{CC} and indeed, it blinks! Of course, people who prefer traditional electronics might comment that the same result could have been achieved more easily using two transistors, but this program does more. It also initialises the serial interface, which now can be used. In order to see that transfers are possible, we need a terminal emulator program.

The simplest approach is to use the Basic terminal program from the course, but the communications parameters must be correct. The C program uses 9600 baud, while Basic.exe normally uses 19,200 baud. However, it is easy to change the transfer rate used by this program. Just open the file Basic.ini with a text editor and add the line 'Baud=9600' (see **Listing 2**).

Listing 2. Content of the modified .ini file for the Basic terminal.

```
[AHBASIC]  
COM=2  
Baud=9600
```

The terminal program will now show what the C program sends, which is a series of '0' and '+' characters that alternate with each change of state of the switched output P1.0 (see **Figure 2**). This is because the program calls the function `putc` to output individual text characters.

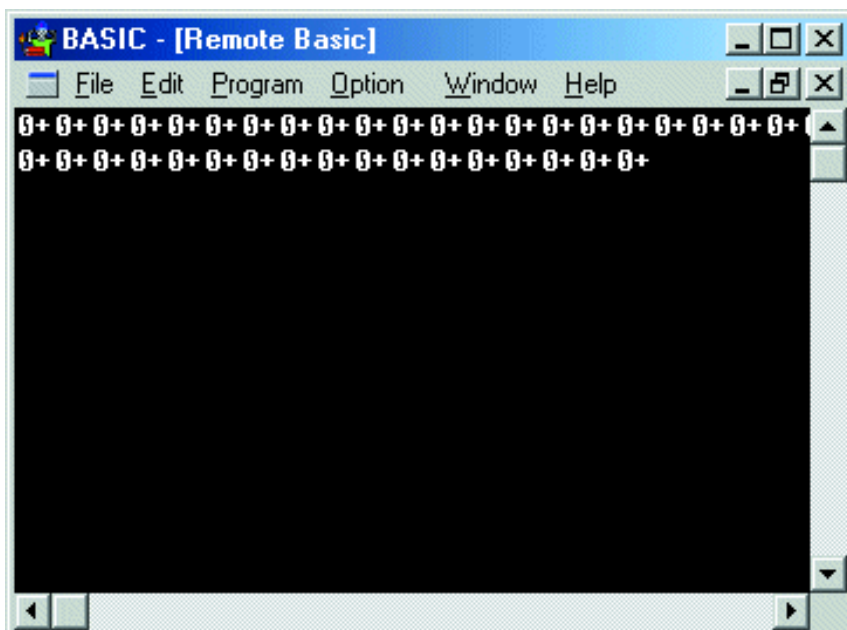


Figure 2. Text outputs in the Editor window.

Fast port outputs

Now that we've seen the first example, it's time to write our own C program. Let's start with a program that simply generates fast port outputs, so that we can make some comparisons with the programming languages we used before.

The first thing to do is to create a new project. A new project name must be entered under Project/New Project. Here we choose the name Output. READS51 now creates a new directory with the name Work\Output. Next, we open a new module under Module/Create Module and give it the name Output. The port output program is shown in **Listing 3**.

The compiler has to know which project it is supposed to translate. This is accomplished by running Project/Set Project Active once. If you forget to do this, the last project that was processed will be translated. The newly generated program code Output.hex can now be transferred to the program memory of the processor using MicroFlash. All that we need to verify this function is an oscilloscope or a set of headphones. The highest-frequency signal will be found on P1.0. It has a frequency of 4 kHz; the period is 250 μ s. The program needs 125 μ s for each new port output.

In contrast to Basic, C allows different types of variables. The first sample program uses a variable n of type int, which means an integer variable with a value range of -32768 to +32767, while the second example uses a variable of type unsigned char, which corresponds to a byte. However, experiments have shown that this does not result in any significant difference in execution speed.

The effectiveness of the individual programming languages can now easily be compared (see **Table 1**). The most important criteria are the amount of memory taken up by the program, its speed and the ability to implement a stand-alone program for the Flash microcontroller. Although C programs use the system RAM, the complete program is located in the Flash ROM alone. This is why a C program starts up again when the voltage is switched on, in contrast to a BASIC-52 program.

A frequency divider in C

In looking for a somewhat more complex task, we remembered the divide-by-20 frequency divider we already wrote in BASIC-52. A direct comparison of the two programs can help us recognise differences in the structure and notation. This program has been given

Listing 3. A program for fast port outputs.

```
// ----- READS51 generated header -----
// module   : C:\Rigel\Reads51\Work\Output\Output.c
// created  : 12:33:17, Friday, November 09, 2001
// -----

#define TRUE  1
#define FALSE 0

#include <sfr51.h> // P1 is defined here

main(){
  unsigned char n;
  // endless loop
  while(TRUE)
  {
    for(n=0; n<256; n++)
    {
      P1=n;
    }
  }
}
```

Listing 4. Divide-by-20 frequency divider.

```
// ----- READS51 generated header -----
// module   : C:\Rigel\Reads51\Work\Count\count.c
// created  : 18:26:23, Monday, November 12, 2001
// -----

#include <sfr51.h>

void pulse(void){
  while(P1_0);
  while(!(P1_0));
}

main(){
  int n;
  n=0;
  while(1)
  {
    while (n<10)
    {
      pulse();
      n=n+1;
    }
    P1_1=1;
    while (n<20)
    {
      pulse();
      n=n+1;
    }
    P1_1=0;
    n=0;
  }
}
```

Table 1. Comparison of the three programming languages

Language	Memory	Loop time	Autostart
Basic-52	8 K ROM, RAM	2500 μ s	Only with EEPROM
READS51 C	>4 K ROM, RAM	125 μ s	yes
Assembler	< 1 K ROM	3 μ s	yes

the name Count (see **Listing 4**).

The listing shows a rather decisive advantage of C as a programming language: the programmer is forced to use a structured style. This makes the program easier to read. Here we have the function pulse, which suspends the progress of the program while waiting for the next positive edge on P1.0. When using a function, it is common to pass in a value and receive another value in return. However, the function pulse does not return any parameter (void = empty), and no parameter is

passed to it. C does not make a distinction between functions and procedures, as is customary in Pascal and Delphi; it has only functions.

The bit variable P1_0 yields either '1' or '0'. As long as the condition following while is true (= 1), a loop is executed. The second loop contains the actual condition in negated form, which is expressed by the exclamation mark '!' (! (P1_0)). The second loop is thus exited when the input level changes from '0' to '1', which means when a positive edge is detected. The main routine

calls the function pulse at two places: once for $n = 0, 1, \dots, 9$ and again for $n = 10, 11, \dots, 19$.

Here again the critical question is, what is the highest input frequency that can be applied without any counting errors? For this test, we used a function generator connected to P1.0 and an oscilloscope connected to P1.1. The measurement yielded an upper frequency limit of 3 kHz. To refresh your memory, BASIC-52 only managed a frequency of 50 Hz, while up to 100 kHz is possible using assembler.

(010208-5)

Literature:

Sencer Yeralan/Helen Emery
*Programming and Interfacing The 8051
 Microcontroller in C and Assembly*
 Rigel Press 2000